UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

COMPUTER CENTRE BULLETIN

Vol. 4, No. 5.
3 May 1971

Editor:
Mrs Sarah Barry

NEW PDP-10 FACILITIES

1. NEW FACILITIES

On Tuesday 27 April 1971, a new version of the command decoder will be implemented, together with supporting software. This gives an expanded command capability, and makes available to the remote terminal user a number of new facilities. Included in these are:

(a) Digital plotting

(b) Support for paper tape I/O on Teletypes

(c) Access to files within other projects

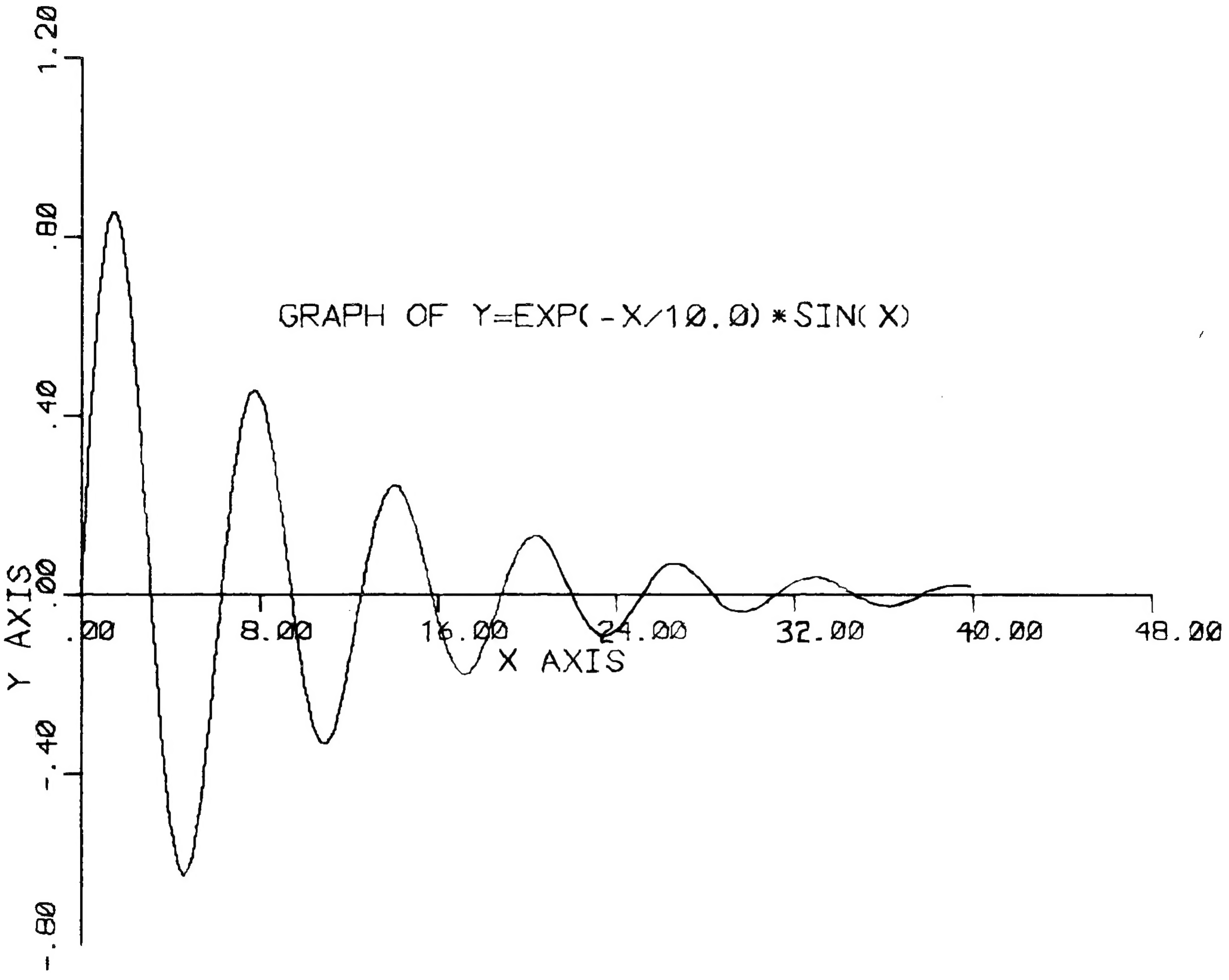
(d) A COBOL compiler

(e) Availability of a number of system programs.

Some further details of these facilities are given below. A revision of the Systems User's Guide and a new manual, giving full details of all the new facilities, are currently in production.

2. DIGITAL PLOTTING

The plotter available on the PDP-10 system is a Model 565 CalComp plotter with an eleven inch wide drum, and a step size of .01 inches. An example of the plotter output is shown below.



PROJECT 16   20-APR-71 15:25

An initial write up, which gives details of the plotter subroutines, plotter operations and use of the plotter, is available from the Centre's Administrative Officer. Full details of the plotter system are included in a new manual which will be printed shortly.

Users are requested to note the following points with respect to the plotter:

(a) To enable users to become familiar with the use of the plotter and the plotting subroutines, the first four weeks of plotter operation will be free of charge. Standard charges for plotting will begin on 24 May 1971.

(b) All plotter output will be in black ink on plain white paper and will be produced by a 0.2 mm rapidograph pen.

(c) Plotter output will be automatically identified by project number. It will not be left in the output shelves, but should be collected from the Centre's Administrative Officer.

(d) The use of the plotter will, for many users, be their first experience of a symbiont (or spooling) operation. The operation is as follows:

    When the command is given to PLOT a data file, that file is not plotted immediately, but is placed in a queue of plot jobs on disk. Subsequently, a symbiont program running under the control of a machine operator runs these plot jobs one at a time producing the graphical output. This mode of operation makes efficient use of the plotter and saves the user time (and money) by not requiring him to wait at his terminal until the plotter is available for his use. Plotted output will normally be available for collection within two hours of termination of the user's job. Details of the PLOT command can be found in section 6.1 (b).

3. EXTENDED COMMAND FORMAT

The concept of a filename has been extended so that files belonging to any directory may be referenced. A *directory* is a file in its own right. The directory name is separated from the filename by a period. For example:

```
directory.filename/processor-program-name
```

refers to filename/processor-program-name in the designated directory.

There are three types of files, each being referenced by a specific type of name, as follows:

(a) User Files

    User file names comprise up to 6 alphanumeric characters, the first being a letter.

examples:

(i)	MYFILE
(ii)	BMDØ2R
(iii)	MYFILE/F4

(b) Project Directories

Project directory names comprise up to 6 numeric characters. The name is the same as the project number of a user and this file is his directory.

example:

124	This refers to the directory of project 124

(c) Special System Files

Special system file names comprise up to 6 alphanumeric characters preceded by a dollar '$'. These filenames refer to input-output devices which can be accessed by the user.

examples:

$ASR	Refers to the paper tape reader and punch on the ASR 33 Teletypes
$TTY	A user's Teletype keyboard and printer

It is intended that any special system filename may be used in a command but, for the moment, it may not be used as the verb in a command.

When users wish to refer to a file in another user's project area, that project's directory name should precede the name of the file.

example:

271.TEST/F4

This refers to the FORTRAN IV file name TEST/F4 within project 271.

If the project directory name is omitted, the standard system files are first searched for a file of the name given, and if this fails the user's own project area is assumed.

examples:

(i) FORTRAN

This references the standard system FORTRAN compiler.

(ii) TEST/F4

This references the file on the user's own project area.

4. SUPPORT FOR PAPER TAPE I/O ON TELETYPES

All I/O using paper tape must use the COPY command. The paper tape reader and punch are referenced by using the directory name $ASR.

examples:

(i)
```
COPY TO=NEWFIL  FROM=$ASR
```
Reads a file from the paper tape reader

(ii)
```
COPY FROM=OLDFIL  TO=$ASR
```
Punches a file onto the paper tape punch

(a) Preparation of tapes

Tapes must be punched on 8-channel paper tape using 7-bit ASCII code with even parity. All records must be terminated by both carriage return (015 ASCII) and line feed (012 ASCII). The tape must be terminated by a control-Z character (032 ASCII). Tapes punched on the remote terminals by the PDP-10 system conform to this standard. No more than one file should be punched on one tape.

(b) Operation of equipment

To feed blank tape on the tape punch:

Turn the Teletype on/off switch to 'LOCAL'
Press the tape punch 'ON' button
Press the keyboard 'HERE IS' button a number of times
Press the tape punch 'OFF' button
Turn the Teletype on/off switch to 'LINE'

If these instructions are not followed, spurious characters will be punched.

(c) Control settings on the reader and punch

The computer will automatically start and stop the reader and punch when it is ready to perform I/O after a COPY command. The user must not switch either reader or punch 'ON' with the Teletype in 'LINE' mode.

Thus, the reader on/off switch should be in the 'STOP' position and the punch 'OFF' button should be down.

5. ACCESS TO FILES WITHIN OTHER PROJECTS

To access a file within another project, quote the project number in the filename.

examples:

(i)
```
COPY 362.HISFIL, MYFILE
```
This copies a file on project 362 to a file on the user's own area

(ii)
```
FORTRAN (LIST) IN=110.SRCFIL BIN=BINFIL LST=LSTFIL
```
The file SRCFIL obtained from project number 110 is compiled, producing relocatable and list files on the user's own area

(iii)
```
21.AFILE/F4
```
This is an automatic compilation and execution of the FORTRAN file obtained from project number 21

The file to be accessed on some other project's directory must have its permission set so that at least READ access to it is allowed.

Permissions are set for two classes of users:

(i) the owner of the project

(ii) all other users (the world)

The permissions are:

FREE	the user can read, write and change permission
WRITE	the user can read and write
READ	the user can read
NONE	the user cannot do anything

For the world, all these permissions are distinct. For the owner, WRITE is equivalent to FREE and NONE is equivalent to READ. The owner can always read his own files and change their permission.

When a file is created, the permission is automatically set to

```
          OWNER=FREE     WORLD=NONE
```

Permissions are set by the PERMIT command. There are now two options to the PERMIT command with assignments OWNER and WORLD.

examples:

(i) `PERMIT(OWNER=READ, WORLD=NONE) FILEA, FILEB(WORLD=READ),FILEC(OWNER=FREE)`

(ii) `PER(R,N) FILEA, FILEB(WORLD=R), FILEC(F)`

For FILEB, READ is assumed for the owner.

For FILEC, NONE is assumed for the world.

6. NEW AND EXTENDED COMMANDS

This version of the decoder provides a number of new commands, together with new options and arguments for some existing commands. These are briefly outlined below. Full details of all these will be included in revision 1 to the System User's Guide.

6.1 New Commands

(a) COBOL

A COBOL compiler is now available on the PDP-10 system. The command to use COBOL is

```
COBOL( BIN    LIST  ,MACRO,MAP)
      NOBIN, NOLIST

{IN=}filename-1,{BIN=}filename-2,{LST=}filename-3
```

filename-1 is the name of the source file
filename-2 is the name of the resulting relocatable file
filename-3 is the name of the list file

Automatic compilation will work using the processor program name CBL.

(b) PLOT

This is used for transmitting plot output files to the plotter symbiont. The command is

```
PLOT filename-1, ..., filename-n
```

filename-1, ..., filename-n are the names of the data files output by the plotter subroutines.

(c) MACRO

MACRO is the assembly language for the PDP-10. The MACRO command is

```
          BIN         LIST
    MACRO(     ,CREF,       )
          NOBIN       NOLIST

    {IN=}filename-1,{BIN=}filename-2,{LST=}filename-3
```

filename-1 is the name of the source file
filename-2 is the name of the resulting relocatable file
filename-3 is the name of the list file

Automatic compilation will work using the processor program name MAC.

(d) COMPARE

COMPARE compares two ASCII files and outputs the differences between the two.

```
             FILE1                FILE2
    COMPARE {     =}filename-1,{     =}filename-2,{LST=}filename-3
             F1                   F2
```

If the LST argument is omitted, the third file appears on the Teletype.

6.2 New Options in Existing Commands

Existing commands

(a) COPY

```
          ASCII
    COPY (     ,COMPRESS)
          BIN
```

ASCII for copying files of ASCII characters

BIN for copying relocatable files

COMPRESS removes sequence numbers and trailing blanks from ASCII records, and converts multiple spaces to tabs.

(b) PERMIT

These details have been given in section 5.

(c) RUN

```
        MAP
    RUN(NOMAP ,DDT)
        SYMBOL
```

This will run the named files with the debugging package DDT.

6.3 New Arguments in Existing Commands

(a) DIRECTORY

The DIRECTORY command can now have an argument string which specifies selective listing of part of a directory.

example:

```
        DIRECTORY ALL/F4
```

This will list all files with the processor program name of F4

(b) FORTRAN

Three arguments can now be used in the FORTRAN command.

```
         BIN        LIST
FORTRAN(     ,CREF,       ,MACRO)
         NOBIN      NOLIST

{IN=}filename-1,{BIN=}filename-2,{LST=}filename-3
```

filename-1 is the name of the source file
filename-2 is the name of the resulting relocatable file
filename-3 is the name of the list file

7. AVAILABILITY OF SYSTEM PROGRAMS

In addition to the system facilities outlined above, the following DEC system programs are now available.

```
BINCOM
CHESS
FUDGE2
LOADER
PIP
RUNOFF
SORT
TECO
```

These programs can be obtained simply by entering the program name. In most cases the program will return an asterisk to the Teletype and wait for the necessary DEC command string to be entered.

Users should not use any of the above names for their own files.

8. SOFTWARE CLASSIFICATION

A statement in the April issue of the Computer Cetnre Bulletin outlined four categories or types under which it was intended to classify all the software available on the PDP-10 system.

Briefly, the categories are as follows:

Type 1 System software that has been formally tested, is documented, and is supported with educational and consulting services.

Type 2 Application programs that have been formally tested, documented and are supported with educational and consulting services.

Type 3 Programs of general interest that satisfy basic standards of testing and documentation. These programs are given some support but this support has low priority.

Type 4 Programs that are made available in the author's original form and have not been tested. These programs attract no support.

The following system programs and facilities have now been classified as Type 1 programs.

(a) System programs and compilers

The monitor
LOGIN
FINISH
Command Decoder
Batch
Editor
FORTRAN
BASIC
FORTRAN library (including the plotter and overlay subroutines)
ACCOUNT
PASSWORD

(b) The facilities available via the following commands

COPY
DAYTIME
DELETE
DIRECTORY
KEEP
OVERLAY
PERMIT
PLOT
RENAME
TIME
TYPE
RUN (excluding the DDT option)

All other programs and facilities at present are classified as Type 4 programs.

LIBRARY ACCESSIONS

BARTON, Richard F.	*A primer on simulation and gaming* 1970 (001.424 BAR Main)
INTERNATIONAL CONFERENCE ON METHODOLOGIES OF PATTERN RECOGNITION Honolulu 1968	*Methodologies of pattern recognition* 1969 (001.533 INT Maths)
VAUGHAN, James A.	*Banking computer style* 1969 (332.1018 VAU Engin)
JACOBSON, David H.	*Differential dynamic programming* 1970 (519.92 JAC Maths)
JOHN, Fritz	*Lectures on advanced numerical analysis* 1967 (517.6 JOH Maths)
KUPPERMAN, Robert H.	*Mathematical foundations of systems analysis* 1969 (517.5 KUP Main)
MITCHELL, Andrew R.	*Computational methods in partial differential equations* 1969 (517.383 MIT Maths)
WEYRICK, Robert C.	*Fundamentals of analog computers* 1969 (621.381957 WEY Phys)
ROTHERY, Brian	*The art of systems analysis* 1969 (658.502 ROT Main)
SYMPOSIUM ON ON-LINE COMPUTING SYSTEMS, University of California, Los Angeles, 1965	*On-line computing systems* 1965 (Qto 651.8 SYM Engin)
TAVISS, Irene, comp.	*The computer impact* 1970 (651.8 TAV Main)

PDP-10 FORTRAN IV

Execution Error

To minimize confusion between the character sets of the 026 and 029 keypunches, it was our intention that both punchings for certain special characters used by FORTRAN should be accepted. Appendix D of MNT-5 FORTRAN IV FOR THE PDP-10 lists the character sets. In particular, the closing parenthesis character ')' was to be represented by the punching 11-8-5 (029 punch) and 12-8-4 (026 punch). Because of an oversight, conversion is not done during execution for this particular character and only the punching 11-8-5 is accepted.

This error will be remedied as soon as practicable, but in the meantime it is suggested that the 029 punching only be used.

Error in the Use of Scratch Files

Under some circumstances, it appears that creation of a binary record after a formatted record can cause selective overwriting of the first formatted record. Users are recommended to avoid the creation of scratch files containing both formatted and binary records.

GE-225 FORTRAN IV

Execution Error

The intermittent error in FORTRAN IV execution which was reported in previous editions appears to have been corrected. Our thanks to those who have patiently cooperated with us in trying to locate this very troublesome problem.

E and F Output Conversion

E and F output conversion routines can give improper results for very small quantities (less than about 1.0E-70). If no scaling factor is used, then F conversion will yield a zero result, but E conversion leads to an improper but small result. If a scaling factor (i.e. 2PF8.2) is used, then the output routine can print a very large number, for example 55555.55.

PROGRAMMING WITH DECISION TABLES

M. J. McLean

Mr. McLean graduated from The City University, London in 1968 with First Class Honours in Civil Engineering. He then went to the University of London Institute of Computer Science where he was awarded an M.Sc. with Distinction in Computer Science. He is currently a Lecturer in the Department of Computer Science at the University of Queensland.

ABSTRACT

It has been found that when decision table facilities are used in programming there is a reduction in the time required for writing and debugging the programs. The use of decision tables has been most popular in the field of commercial data processing and especially as an extension of COBOL.

This paper introduces the use of decision tables in programming and specifically follows the development of the modern software which allows the use of decision tables in COBOL. Finally, it predicts future developments in this field.

INTRODUCTION

Decision tables have been used in programming for over ten years. In that time, they have been introduced into a variety of languages and used in a variety of applications. In most of these cases, the conclusion was that decision tables reduce the time required for writing and debugging programs. However, it is only comparatively recently that they have become generally recognized as a programming tool. One of the main objections to using decision tables in the past was that there was no standard which meant that a company which used them was tying itself to a particular machine.

Most modern decision table software supports an extension of COBOL based on the language DETAB-65. This paper introduces decision tables as a programming tool by outlining DETAB-65. It then discusses decision table translation and extended entry decision tables. Finally, it predicts future developments in this field. It is assumed that the reader is already familiar with the use of limited, extended and mixed entry decision tables outside the field of programming.

OUTLINE OF DETAB-65

DETAB-65 was designed and implemented as a practical foundation on which to experiment with decision table languages. The original proposals were made by the CODASYL Systems Group and the development and implementation was carried out by a branch of the Special Interest Group on Programming Languages (SIGPLAN) of the ACM.

The language was designed to be easy to implement and modify. Also, it was intended for wide distribution, so it needed to be machine independent. For this reason, a pre-processor was written in COBOL to convert DETAB-65 source code to COBOL. In both cases Required COBOL-61 was used so that DETAB-65 could be run with very little modification on any machine with a COBOL compiler. Only limited entry decision tables were initially implemented, although provision was made for later modification to include extended and mixed entry tables.

DETAB-65 is exactly the same as ordinary COBOL except for the actual decision tables which are placed at the end of the PROCEDURE DIVISION. The decision tables are translated into COBOL sections which are also placed at the end of the PROCEDURE DIVISION. Thus, tables cannot be executed as part of the normal in-line code. Instead, they must always be entered by means of a PERFORM or GO TO verb.

	Card Ident	*Stub Area*	*Rules Area*				
		(table name)	*(size of table)*				
Table Header	0 0 1 0 0 0 0	ENROLMENT-SEARCH	004005005				
Rules Header	0 1 0 0 0 1		001	002	003	004	ELS $
Condition Area	0 1 0 1 0 1	COURSE EQ COMP-SC-1	Y	Y	-	-	
	0 1 0 1 0 2	COURSE EQ COMP-SC-E	-	-	Y	Y	
	0 1 0 1 0 3	SEX EQ MALE	Y	N	-	-	
	0 1 0 1 0 4	ATTENDANCE EQ FT	-	-	Y	N	
Action Area	0 1 0 1 0 5	ADD 1 TO C-SC-1-CNT	X	X	-	-	
	0 1 0 1 0 6	ADD 1 TO C-SC-E-CNT	-	-	X	X	
	0 1 0 1 0 7	ADD 1 TO MALE-CNT	X	-	-	-	
	0 1 0 1 0 8	ADD 1 TO FT-CNT	-	-	X	-	
	0 1 0 1 0 9	ADD 1 TO TOTAL	X	X	X	X	

Figure 1. <u>A Decision Table in DETAB-65</u>

The tables are organized in the conventional way with vertical rules (Fig. 1). Each table consists of three areas:

(a) Header Area

(b) Condition Area

(c) Action Area

(a) *Header Area*

This usually consists of two cards, the Table Header and the Rule Header.

The Table Header is the first card of the decision table and is recognized by the pre-processor by '0000' in columns 4-7. In addition to marking the start of the table, this card contains the table name and the number of conditions, actions and rules which the table contains.

The Rule Header is the second card of the decision table. It divides the table vertically into Stub Area and Entry Area, and divides the Entry Area into individual rules. Each rule has a three-digit number and the numbers are sequential starting at 001. The rule numbers do not have any bearing on the order in which the rules are tested. The Stub Area starts at column 9 and can be any number of columns, the area being terminated by the start of Rule 001. Each rule can occupy between 3 and 12 columns (only 3 columns are needed for limited entry tables). The rules start in the first column containing the rule number, and finish at the start of the next rule.

The last rule does not have a number, It is always the ELSE rule (the rule which is satisfied when none of the other rules is satisfied) and is designated by 'ELS'. The ELSE rule is terminated by a '$'.

(b) *Condition Area*

Conditions are written in the conditions stub. A condition is any normal COBOL condition with the IF implied. If the condition is too long to fit onto the line within the condition stub, then a continuation line may be used and the condition is continued in the condition stub of this line. Up to 9 continuation lines may be used, these being recognized by a special mark in one of the first 8 columns of the card.

Each condition entry must be placed in the second column of the rule, and the rest of the rule must be left blank. 'Y' and 'N' are used to indicate 'YES' and 'NO' respectively. A hyphen or blank indicates the 'Don't Care' condition.

(c) *Action Area*

This is similar to the Condition Area. The Action Stub contains actions that are normal COBOL statements. These can be any length, subject to the limit imposed by 9 continuation lines. The action entries contain 'X' in the second column of the rule if the action is to be executed; hyphen or blank if it is to be ignored.

The DETAB-65 language outlined above was very successful and has since become the basis of the software support given by some manufacturers.

DECISION TABLE TRANSLATION

The condition area of each decision table is translated by the pre-processor into a tree structured branching procedure.* A decision table can usually be

* There is a class of translation techniques, the Mask Techniques, which do not generate trees. The code produced by these techniques generally requires less storage than tree structures. However, this code cannot be represented in COBOL source code so these techniques cannot be used in pre-processor systems.

represented by several different but equivalent trees. Algorithms have been published [2 & 3] which select the tree with either minimum average execution time or minimum storage requirement. However, these algorithms are not normally used. Instead, algorithms are selected which are fairly fast in execution and which produce reasonably efficient output.

The DETAB-65 pre-processor checks each decision table for logic errors. If it finds a logic error, it outputs a diagnostic message and ceases to generate source code. The two types of logic error which cause this to happen are Redundancy and Contradiction. After the DETAB-65 had been released, King [4] suggested that it is wrong to cease code generation due to either of these conditions.

Redundancy occurs when two rules represent the same conditions and specify the same actions. This is not a logic error. It does not matter which set of actions is executed when the redundant set of conditions occurs. In Figure 2, rules 1 and 2 contradict since they both represent the condition (Y,Y) and they specify different actions to be performed when this condition occurs. The DETAB-65 pre-processor would require one of the hyphens to be replaced by an 'N', thus removing the contradiction (Fig. 3). However, examination of the conditions in Figure 2 shows that the condition (Y,Y) can never occur. Suppose R1 is the most frequent outcome in the context in which the program is to be used. Then, if the table is translated as it is with the contradiction still present, the tree in Figure 4 will be generated. This tree cannot be generated from the table in Figure 3(a) which requires both rules to be satisfied before R1 holds. It is therefore possible for the unnecessary removal of contradictions to result in the generation of inefficient code.

	R_1	R_2	R_3
Age < 18	Y	–	N
Age > 65	–	Y	N
GO TO	1	2	3

Figure 2. A Decision Table Which Contains a Contradiction

		R_1	R_2	R_3
Age	18	Y	-	N
Age	65	N	Y	N
GO	TO	1	2	3

(a)

		R_1	R_2	R_3
Age	18	Y	N	N
Age	65	-	Y	N
GO	TO	1	2	3

(b)

Figure 3. The Contradiction in Figure 2 Has Been Removed

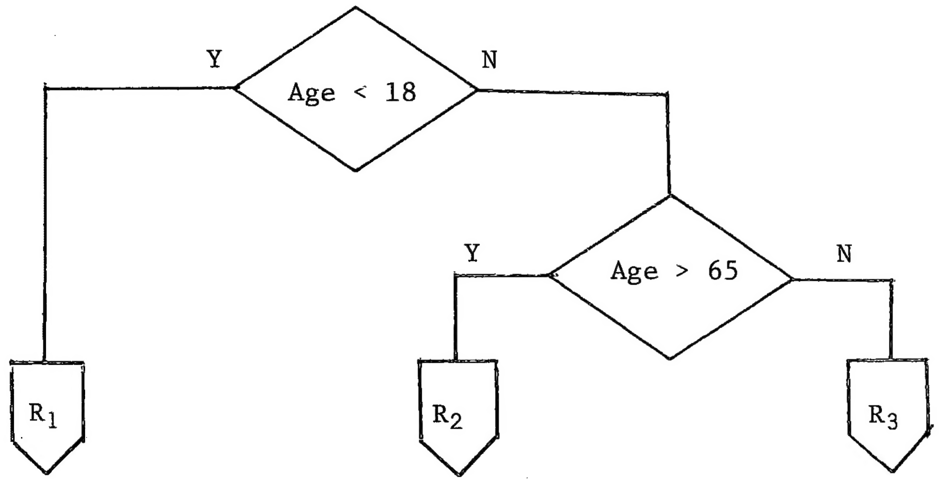


Figure 4. The 'tree' Generated From Figure 2

King proposes that all redundancies should be ignored. Contradictions should be reported as 'possible errors', but these should not interrupt the translation process. The errors should be reported by printing the numbers of the rules which contradict and the combination of entries which is common to both rules. It is up to the programmer to check this printout for real errors.

EXTENDED ENTRY DECISION TABLES

Extended entry decision tables are normally more convenient to program than the corresponding limited entry tables as they are more compact and more easily readable. However, software support for this type of decision table has lagged behind the support for limited entry tables because of the problems of implementation. It is normal for the extended entry table to be translated into a limited entry table which is, in turn, translated in COBOL. The main problem to be resolved is that of finding the correct balance between the ease of programming, the speed of translation and the efficiency of the code which is produced.

At one extreme there is the extended entry decision table which allows all conditions which make sense (and those which don't make sense) to be used. Each condition is simply composed of the contents of the condition stub concatenated with the contents of a condition entry. This combination is simply copied into the generated code without being checked by the pre-processor. Similarly, actions are composed by the concatentation of the contents of the action stub with the contents of an action entry.

This method allows maximum programming flexibility together with high speed translation. Unfortunately, the code produced is extremely inefficient since there has to be a separate condition in the limited entry decision table for every condition entry in the extended entry table.

In order to produce more efficient code, it is necessary for the pre-processor to analyze the conditions and so recognize relations between conditions in the same row of the table. This requires the pre-processor to duplicate some of the work of the compiler and this considerably increases the processing time.

The solution seems to be to allow the use of mixed entry decision tables and to restrict the extended entries to the few most commonly required formats. Four conditions formats and two action formats satisfy most requirements (Fig. 5). In these examples operand means an identifier, a literal or an arithmetic expression.

	Stub	*Entry*
Conditions	Operand Relation	Operand
	Operand	Relation Operand
	(blank)	Condition-name
	(blank)	NOT Condition-name
Actions	PERFORM	Procedure-name
	GO TO	Procedure-name

(a) Formats

A EQ	B
A	EQ B
	PROGRAMMER
	NOT ANALYST
PERFORM	TABEL-1
GO TO	LOOP

(b) Examples

Figure 5. <u>The Most Commonly Required Extended Entry Formats</u>

FUTURE DEVELOPMENTS

Decision tables are becoming increasingly popular as more computer manufacturers offer software support. Most of this support is based on the facilities offered in the DETAB-65 pre-processor. It seems likely that before long this limited entry decision table feature will be included in CODASYL COBOL and USA Standard COBOL (the two COBOL standards). Then COBOL compilers can be expected to appear with decision table features built in. This will increase efficiency and permit space saving Mask methods to be used in the object code.

Later developments are likely to be the introduction of highly flexible mixed entry decision table facilities. These facilities will not then be an extra cost to translare since the translator and compiler will be one program.

REFERENCES

[1] POLLACK, S.L. *Decision Tables Directly into Programs.* Applications of Decision Tables, ed. McDaniel. Brandon/Systems Press Inc. 1970.

[2] REINWALD, D.T. and SOLAND, R.M. *Conversion of Decision Tables to Optimal Computer Programs I : Minimum Average Processing Time.* JACM Vol.13, p.339.

[3] REINWALD, D.T. and SOLAND, R.M. *Conversion of Decision Tables to Optimal Computer Programs II : Minimum Storage Requirement.* JACM Vol.14, p.742.

[4] KING, P.J.H. *Ambiguity in Limited Entry Decision Tables.* CACM Vol.11, p.680.